INFORMATION SYSTEMS SERVICES

White Rose Grid Node 2 (Snowdon) User guide

This is a Getting Started document for new users of White Rose Grid Node 2 (Snowdon). Users wishing to use this Intel-based cluster should read this guide carefully before logging in.

AUTHOR: Dr. A. N. Real

DATE: July 2005

EDITION: 2.2

FREE

1 Introduction ... 3
1.1 About WRG Grid node 2 ... 3
1.2 Becoming a user ... 3
1.3 Connecting, logging into and logging out of the system ... 3
2 Resource allocation ... 4
2.1 Disk Space ... 4
2.2 Usage accounting ... 4
3 Software development environment ... 5
3.1 Compilers ... 5
3.1.1 Example: Compilation and execution of a serial Fortran program ... 5
3.1.2 Example: Compilation and execution of a serial C program ... 5
3.2 Using MPI ... 6
3.2.1 Choosing a particular compiler ... 6
3.2.2 Running a job using the SCore Multi-user environment ... 6
3.2.3 Example: Compilation and execution of an MPI Fortran program ... 7
3.3 The Shell ... 8
3.4 Editors ... 8
3.5 Printing ... 9
3.6 Debugging ... 9
4 Exclusive runtime environment ... 11
4.1 About SGEEE ... 11
4.1.1 SGEEE queues ... 11
4.1.2 Policies for job prioritization ... 11
4.2 Submitting jobs to SGEEE ... 11
4.2.1 Submitting jobs using `qsub` ... 12
4.2.2 The GUI qmon ... 14
4.3 Debugging on main cluster ... 15
4.4 Usage accounting statistics ... 16
5 On-line information ... 16
6 Help and user support ... 16
7 Emailing list ... 17
8 Hints ... 17
8.1 Links* ... 17
9 Acknowledgement ... 17
Appendix A Useful UNIX commands ... 18
Appendix B Compiler quick reference ... 19
Portland group compilers ... 19
Intel compilers ... 19
Compatibility issues ... 20

1 Introduction

This document contains information for users of the White Rose Grid Node 2 service at the University of Leeds. The document explains how to apply for a username and access the system, and gives the necessary information required to start using the service.

This machine is part of the White Rose Grid facilities which are managed jointly with two partners from the White Rose universities: Sheffield and York. The White Rose Grid (WRG) Consortium, which operates under the auspices of the White Rose University Consortium, comprises those researchers from the three White Rose universities whose computational research requires access to leading-edge technology computers.

The WRG equipment has been acquired from, delivered and installed by Esteem Systems plc. together with Sun Microsystems, and Streamline Computing Ltd.

These systems, which are located at the University of Leeds, are operated and supported by the Information Systems Services (ISS) staff on behalf of the White Rose Grid Consortium. Information on using the Node 2 facility is given below; for further assistance pleas contact ISS Helpdesk via email to helpdesk@leeds.ac.uk or telephone 0113 343 3333.

1.1 About WRG Grid node 2

The White Rose Grid Node 2 system is a cluster of Intel servers provided by Streamline Computing. Each machine is based around dual 2.2 or 2.4GHz Xeon processors with 2Gbytes of memory. Gigabit Ethernet and Myrinet 2000 networks serve as the cluster's interconnect, with normal disk traffic traveling via the Ethernet and Myrinet 2000 providing a fast communications backbone for parallel applications. There are 128 compute nodes which are dedicated to batch processing, delivering 256 processors for dedicated batch use. Under normal operation, there are 14 further nodes which can be used for interactive use but these are substituted into batch operation in the event of node failure.

An additional server acts as the front end to the system. It schedules the workload on the cluster nodes and serves important files to the cluster. This machine is known as **`snowdon`**. As only this machine is seen by users, the name **`snowdon`** is interchangeably used to describe the cluster as a whole. Users home directories are served from a shared filestore hosted on the White Rose Grid Node 1 machine.

The systems run an operating system based upon Redhat Linux 7.2. GNU, Portland Group, and Intel compilers are available on the system with parallel applications handled through the **`SCore`** multiprocessor environment. Batch processing capabilities are provided by the Sun Grid Engine, Enterprise Edition product.

1.2 Becoming a user

To register, users are required to complete the ISS Application Form for a computer Username. The completed form must be signed by the WRG node 2 representative and handed in at the ISS Helpdesk.

Note: Once you have been registered, your allocated username and password will be sent to your WRG Node 2 representative.

1.3 Connecting, logging into and logging out of the system

The system is connected to the Leeds University campus Network via a 100Mbit/s Ethernet switch and can be accessed from any networked computer. You can use a variety of terminal types, e.g. workstation, PCs, that support TCP/IP, to connect to the system. The hostname is **`snowdon.leeds.ac.uk`** and the IP address is **`129.11.33.231`**.

It is recommended that secure shell is used to log into the system. To connect from a UNIX platform, issue the command:

```
% ssh –X username@snowdon.leeds.ac.uk
```

where **username** is your username assigned by ISS. Once your connection is established, you will be prompted for a password which is available from your WRG representative. When logged on, you should change your initial password with:

```
% yppasswd
```

To leave the **snowdon** system type:

```
% logout
```

The interactive compute nodes are accessed from a separate front-end system located at **snowflake.leeds.ac.uk** (**129.11.33.247**). This system shares its filestore and all software (compilers, etc.) with the **snowdon** main cluster.

Program development: editing, compiling and debugging; may be undertaken on either **snowdon** or **snowflake**. However, programs can only be launched interactively (using the Score multi-user environment) from **snowflake** and in batch mode (through Sun Grid Engine) from **snowdon**.

2 Resource allocation

The White Rose Grid project is a collaborative venture between the three White Rose universities. A certain proportion of resources are shared between the three institutions. WRG node 2 allocates 75% of resources, equally, between the three shareholding groups; 25% to the WRG of which 5% is allocated to the users of WRG node 1.

2.1 Disk Space

Disk storage for user home directory is provided by Sun StorEdge T3 Fibre Channel disk technology. At present there is one rack with 4 StorEdge T3 disk arrays. Both WRG Nodes 1 and 2 are attached to a shared filestore that provides 2TB of usable disk space.

The storage resource is managed by the SAMFS hierarchical storage management filesystem. This manages files in two storage levels – a cache on disks and an archive on removable media such as tape. Within this filesystem, copies of files on disk are taken for backup and disk space is freed up by automatically moving old files to tape. Consequently, the restoration of deleted files is more convenient than retrieving backup from tape storage.

To ensure efficient job execution at times when the external network is under heavy load and WRG node 1 may be unreachable, a 720GB RAID array is installed locally on Snowdon. This is mounted as scratch file space (under **/scratch**). Users are encouraged to use this directory when undertaking their long-running batch jobs. This has significant performance benefits over using the remote home filestore. Users are reminded that data stored in this directory is not backed-up. You should therefore copy all important data from the **/scratch** area after the run has completed. As there is a limited amount of disk space which is shared between all users, you are requested not to use this area as a general data storage area – all data should be moved or deleted after each run.

2.2 Usage accounting

All CPU usage is recorded and is shown, per shareholding department, in monthly usage accounting reports. Memory use and disk I/O transfers are also recorded and may be reported in the future.

3 Software development environment

The operating system on the **snowdon** cluster is a version of Unix based upon Redhat Linux 7.2. It provides full facilities for scientific code development, compilation and execution of programs. A list of some useful Unix commands are presented in Appendix A.

3.1 Compilers

C and Fortran programs may be compiled using the GNU, Portland Group or Intel compilers. The invoking of these compilers is summarized in the following table:

Language	GNU compiler	Portland compiler	Intel Compiler
C	**gcc**	**pgcc**	**icc**
C++	**g++**	**pgCC**	**icc**
Fotran77	**g77**	**pgf77**	**ifc**
Fortran90/95		**pgf90**	**ifc**
High Performance Fortran		**pghpf**	

All of these commands take the filename containing the code to be compiled as one argument, followed by numerous options. Details of these options may be found through the UNIX **man** facility, e.g. to find details of the options of the Intel Fortran Compiler issue:

```
% man ifc
```

Common options for the Portland and Intel compilers are shown in the tables in Appendix B.

Once compiled, a program can be run by specifying its name on the command prompt.

3.1.1 Example: Compilation and execution of a serial Fortran program

Assuming that the Fortran 77 program source code is contained in the file **mycode.f**, to compile using the Portland group compiler type:

```
% pgf77 mycode.f
```

In this case the code will be output into the file **a.out**. To run this code issue:

```
% ./a.out
```

at the UNIX prompt.

To add some optimization, when using the Portland group compiler, the **–fast** flag may be used. Also **–o** may be used to specify the name of the compiled executable, i.e.:

```
% pgf77 –o mycode –fast mycode.f
```

The resultant executable will have the name **mycode** and will have been optimized by the compiler.

3.1.2 Example: Compilation and execution of a serial C program

Assuming that the program source code is contained in the file **mycode.c**, to compile using the Intel C compiler, type:

```
% icc –o mycode mycode.c
```

In this case, the executable will be output into the file **mycode** which can be run by typing its name at the command prompt:

```
% ./mycode
```

3.2 Using MPI

MPI (Message Passing Interface) is a specification for the user interface to a message passing library used for writing parallel programs. It was designed by a broad group of parallel computer vendors, library writers and application developers to serve as a standard. MPI is implemented as a library of routines which can be used for the development of portable Fortran, C and C++ programs to be run across a wide variety of parallel machines, including massively parallel supercomputers; shared-memory multiprocessors and networks of workstations.

Snowdon uses the SCore system to provide an interface to the compilation and execution of parallel code through the following wrapper scripts:

Language	Wrapper script
Fortran 77	**mpif77**
Fortran 90/95	**mpif90**
C	**mpicc**
C++	**mpiCC**

Please note: It is not necessary to explicitly link against the MPI library, SCore will select the correct one. Also codes should include MPI header files without an absolute path and these header files should never be put into the directory containing the code to be compiled.

3.2.1 Choosing a particular compiler

The above wrapper scripts work on top of the GNU, Portland group and Intel compilers. To specify a particular choice of compiler, simply add the **–compiler** option to the compile line:

Compiler	Option	example
GNU	**-compiler=gnu**	**mpif77 –compiler=gnu file.f**
Portland Group	**-compiler=pgi**	**mpif90 –compiler=pgi file.f90**
Intel	**-compiler=intel7**	**mpicc –compiler=intel7 file.c**

When invoking a particular compiler all the options for optimization etc., for that compiler, may also be specified on the command line.

3.2.2 Running a job using the SCore Multi-user environment

14 compute nodes are configured to run parallel applications under a multi-user environment on the **snowflake** system. Jobs executed in this way will run interactively and all resources will be shared between any other users of the environment. From time to time, not all of the 14 machines will be available, as these compute nodes can be integrated into the exclusive runtime environment in the event of a node failure. To find out if the multi-user environment is running, type:

```
% sctop snowflake
```

at the command prompt. If you are running in a suitable terminal window it should clear and display a parallel "top-like" output showing users jobs running and free nodes. To exit this command press "**Cntrl C**". If the multi-user environment is not running, you will not see any output: Press "**Cntrl C**" for your terminal to respond.

The **scrun** command is used to run parallel code in the SCore multi-user environment. A machine (**comp013**) must be specified that will be the root of the MPI processes through the **scored** variable. To alter the number of processors used, the **–nodes=np** flag (where **np** is the number of processors) may be used, e.g.:

```
% scrun –nodes=2,scored=comp013 ./hello
```

will run the program **hello** using two nodes. Each compute-node contains two processors. To run two processes per compute node, add **x2** to the number of nodes parameter specified to **scrun**, i.e.:

```
% scrun –nodes=2x2,scored=comp013 ./hello
```

will run the program **hello** on 4 processors (2 processes per compute node).

3.2.2.1 Specifying a different communications network

By default SCore is setup to use the most sensible defaults. As Myrinet 2000 offers much better performance for MPI communication over that of Ethernet, the Myrinet network is used by default to handle all communications. To specify that MPI communications proceed via the Ethernet network, the **network=ethernet** flag may be added to the **scrun** command, e.g.:

```
% scrun –nodes=4,network=ethernet,scored=comp013 ./hello
```

will run the program **hello** on 4 processors (one per node) using the Ethernet network.

3.2.2.2 Terminating SCore multi-user environment jobs

To interrupt a program running through the SCore multi-user environment, it is usually sufficient to **press Cntrl-C**. If this is not possible the **pskill** command can be used. Note: this command will kill all jobs owned by the user that have the same name, e.g.:

```
% pskill hello
```

will kill all jobs called **hello** owned by you.

WARNING: the pskill command will delete all processes that include the argument anywhere in the process string. To see all the processes that will be killed by the above command, issue: "**ps –ef | grep hello**".

3.2.3 Example: Compilation and execution of an MPI Fortran program

The program **hello.f** contains the following source code:

```
      PROGRAM hello
      IMPLICIT none
      INTEGER ierr,rank, size
      include 'mpif.h'
      call mpi_init(ierr)
      call mpi_comm_rank(MPI_COMM_WORLD,rank,ierr)
      call mpi_comm_size(MPI_COMM_WORLD,size,ierr)
      print *, ' I am ',rank, 'of ',size
      call MPI_FINALIZE(ierr)
      end
```

To compile this code using the Portland group compiler with optimization, type:

```
% mpif77 –compiler=pgi –fast –o hello hello.f
```

This will produce an executable called **hello**. To run this code, using 4 processors (running one process per node) and communicating over Myrinet issue:

```
% scrun –nodes=4,scored=comp013 ./hello
```

at the command prompt. The output should be as follows:

```
SCore-D 5.8.2 connected (jid=11).
```

```
<0:0> SCORE: 4 nodes (4x1) ready.
  I am            0 of            4
  I am            3 of            4
  I am            2 of            4
  I am            1 of            4
```

To run this code, using 8 processors (two processes per node) instead, issue:

```
% scrun –nodes=4x2,scored=comp013 ./hello
```

at the command prompt. The output should be as follows:

```
SCore-D 5.8.2 connected (jid=12).
<0:0> SCORE: 4 nodes (4x2) ready.
  I am            0 of            8
  I am            3 of            8
  I am            2 of            8
  I am            1 of            8
  I am            7 of            8
  I am            5 of            8
  I am            6 of            8
  I am            4 of            8
```

3.3 The Shell

The **csh** (C-shell) is the default shell on the cluster (although this is replaced by the enhanced, completely compatible **tcsh**). For this shell the basic setup file is called **.cshrc**. Should you wish to change the basic behavior of this shell, change the **.cshrc** file. A **csh** executes the **.login** file and then the **.cshrc** file when you log in and the **.logout** file whey you log out. These files are located in your home directory, to see them type:

```
% ls –la
```

In scripts the C shell is invoked by the following sequence in the first line:

```
#!/bin/csh
```

To setup the environment variables which control the shell's behavior type:

```
% setenv variable_name value
```

where the **variable_name** is the name of the environment variable, and **value** is the value it is to be set to.

The shell is documented in the man pages; type **man csh** (or **tcsh**) to get more details of this shell.

3.4 Editors

The following Unix editors are available on the system:

Program	Description	Command
Vim	Enhanced version of the "vi" terminal editor	**% vi**
Emacs	GUI and terminal editor	**% emacs**
NEdit	GUI editor	**% nedit**

All these editors, when being invoked, allow a filename to be specified for opening. E.g.

```
% nedit file.f
```

will open the file **file.f** for editing.

3.5 Printing

You may print to any of the ISS printers by typing:

```
% lpr –Pprinter_name file_name
```

where **printer_name** is the name of the ISS printer and **file_name** is the file to be printed. The **lpq** command shows the status of a printer, e.g. the list of jobs in a queue.

3.6 Debugging

The Distributed debugging tool (DDT) is available on Snowdon for analyzing the execution of parallel MPI code. This is available on both the development cluster (up to 14 nodes), using the SCore multiuser environment on **snowflake**, and on the main **snowdon** cluster (up to 128 nodes) via Sun Grid Engine and the Score exclusive runtime environment. Compilation on the front end and invoking the debugger on the development cluster will be described here. For details on debugging on the main cluster, please see section 4.3.

To specify that the compiler should generate debugging information, the "**-g**" option should be given to the MPI wrapper script (**mpif77**, **mpicc** etc.) in addition to any other compiler flags; e.g. for the Intel compiler:

```
% mpif77 –compiler intel7 –o problem –g problem.f
```

will compile the program "**problem.f**" and produce an executable "**problem**" that is capable of generating debugging information.

This program can be tested by running it with **scrun**, e.g. to run on four processors, issue:

```
% scrun –nodes=2x2,scored=comp013 ./problem
```

The debugging environment is set up to run on the development cluster where the resources are shared amongst interactive users and those debugging code, thus drops in performance may be encountered when running through this environment compared to that experienced when running exclusively through the batch queues.

To invoke DDT, issue the command "**ddt**" or "**$DDT**" at the command prompt. Once the startup screen has disappeared DDT will display the '**Session control**' dialogue. If the title of this window reads, '**Session Control (queue submission mode**)', click on '**Change**' to select a different MPI implementation and unselect the '**Submit job through queue**' button. The window will hide all of the queue options. Click "**Ok**" to continue. The "**Session Control**" should then be displayed, suitable for use on the development cluster.

To choose the program name, either enter it directly into the '**Application**' box or use the '**…**' button to select the program using the file tree. Arguments to your program can be specified in the '**Arguments**' box. Flags you need to pass to **scrun** (excluding your program name/arguments) can be specified in the '**MPIRun Arguments**' box without the leading "**-**". E.g. to specify that 8 processes should be started, by running two processes on 4 nodes the following should be entered into the **MPIRun Argument**s box:

```
scored=comp013,nodes=4x2
```

The number of processes to start can then be selected in the '**number of processes**' box. This should directly correspond to that specified through the "**nodes=**" flag in the **MPIRun Arguments** box.

The MPI implementation should be set to '**SCore**'. If it is not, select '**change**'. In the new dialogue window select the correct MPI implementation (SCore) and click **Ok**.

At the time of writing, the GNU (**gdb**) and Portland (**pgdbg**) debugging interfaces are available for debugging MPI code, the choice of debugging **interface** should match the compiler that built the executable that will be debugged.

The program will be loaded into DDT when you click '**Run**'.

For further details on using DDT, please see the User Guide provided by Streamline Computing, available either via the web at: http://www.leeds.ac.uk/iss/wrgrid/Documentation/ or on Snowdon in **`/usr/local/ddt/doc.`**

4 Exclusive runtime environment

To ensure the effective use of WRG Node 2 resources, the resource management system, Sun Grid Engine, Enterprise Edition (SGEEE), is installed. The job manager allows system resources to be allocated in a controlled manner to submitted requests. With the exception of very short tests, which can be undertaken through the SCore multi-user environment, all production- and long development-code runs should be executed in this manner to ensure that users have exclusive access to the resources requested.

4.1 About SGEEE

The Sun Grid Engine, Enterprise Edition product is a resource management tool which might be used to enable grid computing. This is a complex and powerful package. Grid Engine is an advanced batch processor that schedules jobs, submitted by users, to appropriate systems available under its configuration according to the resource management policies accepted by the organization. It manages global resource allocation (CPU time, memory, desk space) across all systems under its control. SGEEE controls the delivery of computational resources by enforcing policies set by the administrator.

4.1.1 SGEEE queues

Batch and interactive jobs may be submitted to SGEEE. All jobs submitted to SGEE, will be held in a spool area waiting for the scheduling interval when the scheduler dispatches jobs for processing on the basis of their ticket allocations. Tickets are used to enforce scheduling policies. The more tickets the job is assigned the more important the job is and it is dispatched preferentially. Jobs accumulate tickets from all policies. If no tickets are assigned to the policy then the policy is not used. At each scheduling iteration the number of tickets owned by each job, including the executing jobs, is re-evaluated. Jobs currently executing are also evaluated at each scheduling period and their allocation of tickets may be amended. Tickets assigned by the administrator enable the scheduler to determine which jobs should be run next.

4.1.2 Policies for job prioritization

There are four policies that can be applied to schedule user's jobs. These are as follows:

- Share based (also called share tree) – when this policy is implemented, users are assigned the level of service according to the share they own and the past usage of resources by all users and their intended use of systems. It allows for share entitlements to be implemented in a hierarchical fashion.
- Functional – when this policy is implemented users are assigned the level of service according to a share they own and the current presence of other jobs. This policy is similar to the share-based policy but does not consider the past usage of the system. It also allows for share entitlements to be implemented in a hierarchical fashion.
- Deadline – this policy assigns high priority to certain jobs that must be finished before the deadline.
- Override – this policy requires the administrator of SGEEE to modify manually the automated policy(ies) to priorities vital jobs. It is to be employed in most exceptional circumstances.

The first three policies are managed through the concept of tickets which, like shares, might be assigned to projects, departments, and/or users. The last policy is managed manually by the administrator.

The share-tree policy is adopted to manage the WRG Node 2 resource allocations.

4.2 Submitting jobs to SGEEE

Users submit jobs to a queuing system and the scheduler allocates them to the relevant queue. In order to request specific resources, a script is used which contains a list of options together with commands which are run whilst in the execution environment. There are two ways in which users can submit jobs to this batch system:

- Using the **`qsub`** (command line interface)

- Or using **qmon** (a GUI interface).

Jobs can only be submitted to the batch queues from **snowdon**.

4.2.1 Submitting jobs using qsub

The general command to submit a job with the **qsub** command is as follows:

```
% qsub [options] script_file_name [--script-args]
```

where **script_file**_name is a file containing commands to executed by the batch request.

Commonly used **options** are:

Option	Description
-l h_rt=hh:mm:ss	The wall clock time. This parameter must be specified, failure to include this parameter will result in the error message: "Error: no suitable queues".
-l h_vmem=memory	Sets the limit of virtual memory required (for parallel jobs per processor).
-P project_name	Specifies the project this job is assigned. If you do not specify this parameter your job will run under and be accounted to your default project. **project_name** can be one of: **WhiteRose**, **ISS**, **Computing**, **Maths** or **Earth**
-help	Prints a list of options
-pe score np	Specifies the parallel environment to be handled by the Score system. **np** is the number of nodes to be used by the parallel job. Please note: this is always one more than needed as one process must be started on the master node, which, although does not carry out any computation, is necessary to control the job.
-cwd	Execute the job from the current working directory; output files are sent to the directory form which the job was submitted, not to the user's home directory.
-m be	Send mail at the beginning and at the end of the job to the owner
-S shell	Use the specified shell to interpret the script rather than the C shell (default).
-masterq master.q	Specifies the name of the master scheduler as the master node (**master.q@snowdon.leeds.ac.uk**)
-V	Export all environment variables to all spawned processes.

Alternatively, users may prefer to include some of these options in their execution script rather than on the command line. To specify options that are interpreted by the queuing system within the job submission script, the relevant lines must begin with the sequence "**#$**"; e.g. to declare the name of the **masterq**, specify that the current working directory should be used and ensure that all environment variables are exported to all spawned processes, the following lines may be included in the job submission script:

```
#$ -masterq master.q
#$ -cwd –V
```

4.2.1.1 Serial job submission

Serial jobs may be submitted directly to the queuing system. To run the program **benchmark**, requesting one hour of runtime, the following script may be used:

```
#$ -l h_rt=1:00:00
```

```
#$ -cwd
#$ -V
./benchmark
```

This script can be submitted to the queuing system using the **qsub** command directly. If the above script was contained in the file **run_serial.csh** this would be done by typing:

```
% qsub ./run_serial.csh
```

at the command prompt.

During batch request submission, the script file is spooled so that subsequent changes to the original file do not affect the queued batch request. When your batch request has been submitted successfully to SGEEE, you will receive a message in the format of:

```
Your job 321 ("run_serial") has been submitted.
```

Please note that you will not be able to submit your job to a named queue because the queue is determined by the resources requested by you and all jobs are waiting in a spooling area before the scheduler dispatches them to the queue. If you omit the number of processors you require, your job will be executed on a single processor.

4.2.1.2 MPI job submission

MPI jobs are submitted to the queuing system and handled through SCore. The option "**masterq master.q**" should be present in the job submission script. Furthermore, as SCore handles the parallel run the program must be executed in a special way. This is done through the **scout** command which launches a SCore remote shell runtime environment on hosts assigned from SGEEE. The final option on the **scout** line, which follows a "**–e**", is the command to be run on the remote hosts. This is normally a version of the **scrun** command created for running the job. Following this command, all normal options to **scrun** may be specified. A typical parallel job submission script, **run_mpi.csh**, that runs the program **myprog** will therefore consist of the following:

```
#$ -masterq master.q
#$ -cwd –V
#$ -l h_rt=12:00:00
scout –wait –F $HOME/.score/ndfile.$JOB_ID –e /tmp/scrun.$JOB_ID \
 -nodes=$((NSLOTS-1))x1 ./myprog
```

qsub is used to submit the job to SGEEE, together with the **"-pe score np**" flag, where **np** specifies the number of processors (slots). As SCore always spawns MPI jobs from the front end server, an extra "slot" must be specified to account for this. Thus, to run a parallel job on 4 compute nodes, 5 slots should be specified. i.e. to submit the above script to SGEEE, running on four nodes the following should be used:

```
% qsub –pe score 5 ./run_mpi.csh
```

As all options to **scrun** may be specified "**x1**" may be changed to "**x2**" to enable two processes to be started per node. In the example presented above, this change will cause an 8 processor job to be executed.

Please note: in order that SGEEE can control the batch job, the "**-l h_rt**" flag must be specified. If this parameter is not passed to SGEEE, the job will not be scheduled to run and the error "error: no suitable queues" will be returned. The example above will request 12 hours of runtime.

4.2.1.3 Job output

For each batch job, two files, of the form **jobname.oxxx** and **jobname.exxx** will be produced (where **xxx** is the number of the batch job); these files contain all the standard output and standard error messages that would be normally printed to the screen, respectfully.

If the **–cwd** option is specified then output files are sent to the directory from which the job was submitted, otherwise they will be dispatched to the user's home directory.

To receive email after a batch request has ended execution, the **-m e** option should be specified either in the job script or on the **qsub** command line.

4.2.1.4 Querying queues

The **qstat** command may be used to display information on the current status of Grid Engine jobs and queues. The basic format for this command is:

```
% qstat [switches]
```

Important switches are as follows:

Switch	Action
-help	Prints a list of all options.
-f	Prints a summary of all queues.
-U username	Displays status information with respect to queues to which the specified users have access.

The switches are documented in the man pages; for example, to check all options for the **qstat** command type:

```
% man qstat
```

4.2.1.5 Job deletion

To delete your job issue the following command:

```
% qdel jobid
```

where **jobid** is a number referring to the specified job (available from **qstat**). To force action for running jobs issue the following command:

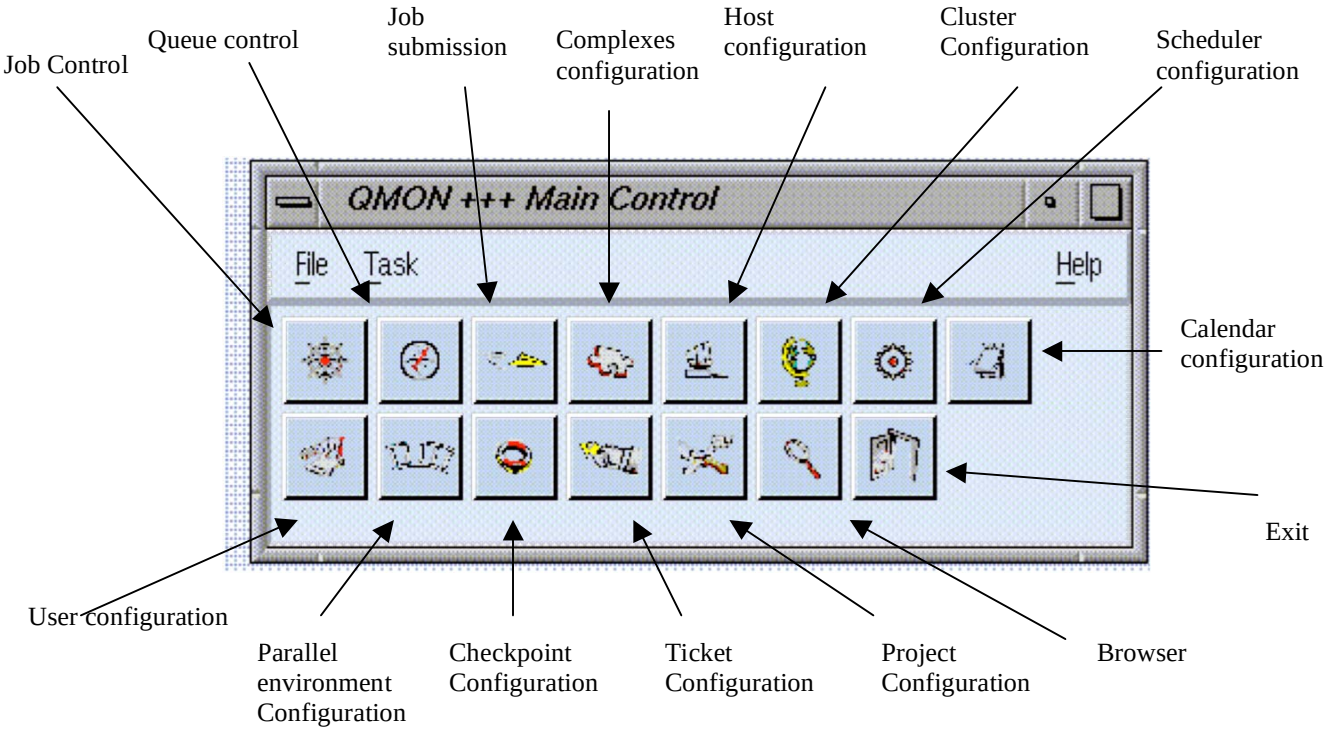
```
% qdel –f jobid
```

4.2.2 The GUI qmon

SGEEE can be used via a Graphical User Interface (GUI) which may simplify the use of the queuing system. The GUI, which runs in X-windows, is called by typing the command:

```
% qmon
```

at the command prompt. The main control window will offer you a number of buttons to submit your job, to query its status, to suspend its execution or to remove it altogether.



4.3 Debugging on main cluster

DDT can interface with Sun Grid engine to facilitate debugging on the main **snowdon** cluster. Users wishing to debug on the development cluster should follow the instructions in section 3.6.

First ensure that the code that will be debugged has been built using the correct compiler options for the executable to contain debugging information.

To invoke DDT, issue the command "**ddt**" or "**$DDT**" at the command prompt. Once the startup screen has disappeared DDT will display the '**Session control**' dialogue.

Click on '**Change**' to select a different MPI implementation and select the '**Submit job through queue**' button. The window containing all of the queue options should then appear. Ensure that the '**MPI implementation**' is set to '**SCore**'.

Firstly a queue template file should be given. It is possible for you to create your own, however several templates have been provided in the **/usr/local/ddt/templates** folder on Snowdon:

6h_sge.qtf: requests 6 hours of runtime,
48h_sge.qtf: requests 48 hours of runtime,
120h_sge.qtf: requests 120 hours of runtime.

The **Submit command** should be set to '**qsub**'
The **Regexp for job id** should be set to '**id\s(.+)\ssubmitted**'
The **Cancel command** should be set to '**qdel JOB_ID_TAG**'
The **Display command** should be set to '**qstat –u <username>**', where **<username>** is your username.
Also select **Template uses: NUM_NODES and PROCS_PER_NODE** and set the number of **PROCS_PER_NODE** to either **1** or **2** (depending upon how many you wish to launch).

Once the above parameters are entered there should be no reason to type them in again, click "**Ok**" to continue. The "**Session Control (queue submission mode)**" dialogue box should then be displayed.

To choose the program name, either enter it directly into the '**Application**' box or use the '**…**' button to select the program using the file tree. Arguments to your program can be specified in the '**Arguments**' box. There should be no need to specify any further variables.

The number of processes to start can then be selected in the '**number nodes**' box, bearing in mind that there will be either 1 or 2 processes started on each node. Unlike when running directly through the batch queues, there is no need to specify an extra runslot/node on the master node as DDT will take care of this for you; i.e. to run on 8 processors using 2 processes per node, set the number of nodes to 4.

Ensure that the MPI implementation is set to '**SCore**'.

At the time of writing, the GNU (**gdb**) and Portland (**pgdbg**) debugging interfaces are available for debugging MPI code, the choice of debugging interface should match the compiler that built the executable that will be debugged.

A batch job will be submitted to SGE when you click '**Submit**' and when scheduled the program will be loaded into DDT.

Please note that due to licensing restrictions on the Portland debugger, only 43 processors in total can be debugged at any one time on Snowdon under pgdbg. On other compilers/interfaces there is no upper limit.

For further details on using DDT, please see the User Guide provided by Streamline Computing, available either via the web at: http://www.leeds.ac.uk/iss/wrgrid/Documentation/ or on Snowdon in **/usr/local/ddt/doc/.**

Please note that when using the batch queues, the resources necessary to run the job may take some time to become available and therefore the program that is submitted via the debugger may stay queued (**qw**) for quite some time. Furthermore within the directory that DDT was launched from, the job output and error files will be hidden with a **.ddt** prefix, you may want to delete these periodically as they do tend to accumulate.

4.4 Usage accounting statistics

The SGEE system is set up to generate accounting statistics for jobs run under this product. This information will is available via the Web at http://www.leeds.ac.uk/iss/wrgrid/Usage/.

5 On-line information

There are various forms of on-line information available. UNIX on-line man pages may be accessed by typing:

```
% man topic
```

Further and more up to date documentation is also provided via the web at:
http://www.leeds.ac.uk/iss/wrgrid/Documentation/.

6 Help and user support

General user queries and further guidance on the use of the system may be obtained via email to helpdesk@leeds.ac.uk. This is the preferred way of dealing with user queries. However, users who require direct user-support may arrange it by sending email to helpdesk@leeds.ac.uk.

7 Emailing list

All new users are subscribed to an emailing list for users of the facility. This is a moderated list and users are encouraged to use it as a discussion forum for problems common to high performance computing and grid technology areas.

To disseminate information to this list please send email to:

wrg-snowdon-users@leeds.ac.uk

8 Hints

- All users are strongly advised to checkpoint their long running programs.
- The simplest way to gather basic data about program performance and resource utilization is to use the time(1) command or in csh the set time command. When running through the exclusive runtime environment the time command should follow the "`-e`" option, before the invocation of `scrun`.

*8.1 Links**

The following links might be useful:

A listing of parallel computing sites maintained by David A. Bader	http://computer.org/parascope/
Edinburgh Parallel Computing Centre	http://www.epcc.ed.ac.uk/
CSAR high performance service at Manchester Computing	http://www.csar.man.ac.uk/
Sun Grid Engine	http://www.sun.com/software/gridware/ http://gridengine.sunsource.net/
SCore	http://www.pccluster.org/
Message passing interface forum	http://www.mpi-forum.org/
The MPI standard	http://www-unix.mcs.anl.gov/mpi/
Designing and building parallel programs by Ian Foster	http://www-unix.mcs.anl.gov/dbpp/
School of computing module: practical parallel programming	http://www.comp.leeds.ac.uk/so31/
Portland group	http://www.pgroup.com/
Intel compiler	http://developer.intel.com/software/products/compilers/

* Please note that these links are provided for convenience only; the author or this page or ISS do not necessarily endorse the views or products mentioned in them.

9 Acknowledgement

Many of the details found in this guide have been adapted from the *Getting Started guide* for Leeds Grid Node 1 document, by Dr J.G. Schmidt.

Appendix A Useful UNIX commands

apropos	Displays the main page name section number and a short description for each man page whose NAME line contains keyword
cat	Reads each file in sequence and writes on the standard output
cd	Changes working directory
cmp	Compares two files
cp	Copies the contents of **source_file** to the destination path named by **target_file**
diff	Compares the contents of **file1** and **file2** and writes to standard output a list of changes necessary to convert **file1** to **file2**
exit	Terminates process with status.
finger	Displays in multi-column format information about each logged-in user, e.g. username and login time
grep	Searches text files for a pattern and prints all lines that contain that pattern
history	View a list of the last commands
jobs	Shows your background jobs
kill	Sends a terminate signal to a process, not necessarily killing the process
ls	Lists the contents of a directory
man	Displays information from the reference manuals
mkdir	Creates a new directory
more	Displays the contents of a text file on the terminal, one screen at a time
mv	Renames (or moves) a file
yppasswd	Changes the password
pwd	Returns the current working directory
qsub	Submits batch jobs to the Grid Engine queuing system
qstat	Shows the current status of the available Grid engine queues and the jobs associated with the queues
qdel	Provides a means for the deletion of one or more jobs
rm	Removes (delete) files
rmdir	Removes (delete) directories
spell	Checks a file for spelling mistakes
tar	Archives and extracts files to and from a single file
write	Reads lines from a user's standard input and writes them to the terminal of another user

Appendix B Compiler quick reference

Portland group compilers

Official documentation for the Portland group compiler may be obtained from the Portland group website.
The Portland group compilers are invoked by the following commands:

Command	Compiler
pgf90 [opts] file.f90	Fortran 90
pgf77 {opts] file.f	Fortran 77
pgcc [opts] file.c	C
pgCC [opts] file.C	C++
pghpf [opts] file.hpf	High performance Fortran

Full documentation is found in the respective **man** pages, however common options include:

Option	Effect
-c	Compile, do not link.
-o exefile	Specifies a name for the resulting executable.
-g	Produce debugging information (no optimization).
-Mbounds	Check arrays for out of bounds access.
-fast	Full optimization with function unrolling and code reordering.
-Mvect=sse	Turn on streaming SIMD extensions (SSE) and SSE2 (PIII/4/Xeon specific optimization) instructions.
-Mvect=prefetch	Generate prefetch instructions.
-tpp p7	Specifies target platform to be Pentium 4.
-g77libs	Link time option which allows object files generated by g77 to be linked in to programs (note: may cause problems with parallel libraries)

Intel compilers

Official documentation for the Intel compiler may be obtained from the Intel website.

The Intel compilers are invoked by the following commands:

Command	compiler
ifc [opts] file.f90	Fortran 95/90/77
icc {opts] file.c	C/C++

Full documentation can also be found in **/opt/intel/compiler70/docs/** with **for_ug.lnx.pdf** giving information for Fortran users and **c_ug_lnx.pdf** for C. Common options are:

option	Effect
-c	Compile, do not link.
-o exefile	Specifies a name for the resulting executable.
-g	Produce debugging information (no optimization).
-C	Checks for nil pointers/allocatable array references at runtime (-CA); check arrays for out of bounds access (-CB); checks for consistent shape of intrinsic procedure (-CS); checks for variables that have not been initialized (-CU) and tests correspondence between subprogram arguments and dummy arguments expected.
-O3	Full optimization with function unrolling and code reordering.
-axW	Generates, on a single binary, code specialized to the Pentium 4/Xeon.
-xW	Generates specialized code to run exclusively on the Pentium 4/Xeon.
-prefetch	Generate prefetch instructions.
-tpp7	Specifies target platform to be Pentium 4/Xeon.
-parallel	Enables the auto-parallelizer to generate multithreaded code for loops that can be safely executed in parallel.

-openmp	Allows the parallelizer to generate multithreaded code based on OpenMP directives.

Compatibility issues

The Intel compiler requires several runtime libraries which are not compatible with the standard GNU libraries under Linux. Below are details of several libraries which may be needed to link your code with the Intel compiler:

Shortcut option	Library	Description
-C90	**libCEPCF90.a**	Link with alternate I/O library for mixed output with the C language.
-posixlib	**libPOSF90.a**	Enable linking with POSIX library.
-Vaxlib	**libPEPCF90.a.**	Enable linking with portability library (very important – contains functions like **getarg**).
-parallel / -openmp	**-libguide.a**	Link with the OpenMP library.

Programs compiled with the Intel compiler will automatically link with the Intel math library (**libimf.a**), in addition to the GNU library (**libm.a**).